Article

VaR Estimation with Quantum Computing Noise Correction Using Neural Networks

Luis de Pedro *, Raúl París Murillo, Jorge E. López de Vergara, Sergio López-Buedo and Francisco J. Gómez-Arribas

Escuela Politécnica Superior, Universidad Autónoma de Madrid, 28049 Madrid, Spain; raul.paris.murillo@gmail.com (R.P.M.); jorge.lopez_vergara@uam.es (J.E.L.d.V.); sergio.lopez-buedo@uam.es (S.L.-B.); francisco.gomez@uam.es (F.J.G.-A.)
* Correspondence: luis.depedro@uam.es; Tel.:+34-91-497-22-52

Abstract: In this paper, we present the development of a quantum computing method for calculating the value at risk (*VaR*) for a portfolio of assets managed by a finance institution. We extend the conventional Monte Carlo algorithm to calculate the *VaR* of an arbitrary number of assets by employing random variable algebra and Taylor series approximation. The resulting algorithm is suitable to be executed in real quantum computers. However, the noise affecting current quantum computers renders them almost useless for the task. We present a methodology to mitigate the noise impact by using neural networks to compensate for the noise effects. The system combines the output from a real quantum computer with the neural network processing. The feedback is used to fine tune the quantum circuits. The results show that this approach is useful for estimating the *VaR* in finance institutions, particularly when dealing with a large number of assets. We demonstrate the validity of the proposed method with up to 139 assets. The accuracy of the method is also proven. We achieved an error of less than 1% in the empirical measurements with respect to the parametric model.

Keywords: neural network; qubit; quantum computing; Monte Carlo; value at risk (*VaR*)

MSC: 81P68

1. Introduction

International regulations compel financial institutions to allocate loss provisions for the investments they made. The criterion for determining the budget allocated to loss provisions is to estimate the investment risk so that the probability of loss remains below a certain threshold. This threshold is usually determined by regulatory authorities, and typically ranges from 1% to 5% [1–3]. For this purpose, value at risk (*VaR*) [4] is a widely used metric for risk management, as it calculates the maximum potential loss within a given probability range, typically set at 99%. For example, if an investment portfolio has a 1-day 99% *VaR* of USD one million, it means that there is a 1% probability that losses after one day will be USD one million or more.

Monte Carlo simulation is one of the available algorithms and methods used in finance to estimate investment risk metrics, particularly the *VaR*. A random number generator is used to create several samples according to a given distribution. These samples are used to calculate a possible result (gain/loss) scenario for a certain asset. Then, the process is repeated with a new set of samples to calculate another result scenario. If these calculations are repeated enough times, we can accurately estimate the distribution function of the random variable that corresponds to the results of the investment.

Traditionally, log-normal, Pareto and other distributions have been used to model financial markets. The log-normal distribution is also relatively easy to use, so it has been extensively applied in recent years to *VaR* estimations [5]. Moreover, it has the advantage of modeling long tails in the distributions. Once the distribution function for the result of a

certain asset has been approximated, the random variable corresponding to the result of the whole investment portfolio can be modeled by accumulating the distributions of all assets. The more samples generated for obtaining the distribution functions of the assets, the more accurate the model of the investment result. However, this approach experiences two challenges.

The first challenge is the growing number of samples needed when the number of assets in the portfolio increases. The computation time limits the use of the Monte Carlo method when the portfolio is large. Several solutions have been used to address this issue, such as the use of high-performance computers, which is a common solution but expensive. Another approach is to combine the asset distributions from the recorded results and calculate the *VaR* by using the resulting theoretical distribution. This approach is called parametric estimation, and it is explained in the following section. The problem is that the complexity of financial markets makes this approach very rigid. Investors feel that the resulting *VaR* is sometimes excessively conservative. This leads to the next challenge.

The second challenge, as mentioned, is the need for "real" random numbers to capture the somehow chaotic environment in which investments are conducted. Quantum computing (QC) can help in this task, given its intrinsic randomness property. As it is based on physical measurements of quantum states, there are no algorithms behind. It is expected that a physical system can model the investment market much better than a pseudo-random number generation algorithm. However, the use of quantum computing to estimate risk, although promising, is affected by the noise inherent to current quantum computers.

In fact, quantum computing is still at an early development stage. Preskill coined the term noisy intermediate scale quantum (NISQ) [6] to describe the quantum computing technology that is nowadays becoming available. That is, quantum computers featuring a limited number of qubits (in the order of tens) and, more importantly, suffering severe noise problems. Noise may jeopardize the usefulness of NISQ technology to solve real-world problems. However, noise-mitigation techniques based on complex neural networks and, more specifically, deep learning methods, have gained interest in recent years.

Addressing these challenges, the contributions of this paper are manifold:

1. We present a method to summarize market behavior with a single Gaussian distribution. This addresses the problem of many assets in a portfolio.
2. A comparison with present parametric and Monte Carlo estimations is discussed. Therefore, the risk of a portfolio with an arbitrary number of assets can be afforded.
3. We show that, using the above-mentioned Gaussian and a Taylor series expansion, *VaR* can be calculated using quantum computing.
4. We present a method to mitigate the noise effect in actual NISQ quantum computers by using neural networks.

The rest of the paper is organized as follows. First, we present the current approach to *VaR* estimation and Monte Carlo methods. Next, we develop the single Gaussian estimation method, and we apply it to actual assets. Then, we propose to use quantum computing to improve the randomness of the solution. After this, we use neural networks to mitigate the noise in the quantum circuit, and discuss the obtained results. Later, we provide a comparison of our approach with other works. Finally, we conclude the paper, highlighting the main results and future works. Different values of *VaR* are used in the examples along the paper to facilitate the visualization of each case.

2. Monte Carlo *VaR* Estimation

The Monte Carlo method for estimating the *VaR* is based on the generation of Gaussian samples with zero median and unitary variance, which are then combined to generate predictions of market values. This method has been extensively used in finance for estimating risks in investments [7]. The objective is to predict the risk for the following investment cycle, usually one day, using historical data. For instance, you have the current values of the assets and the history of previous values for the past months, and you want to predict the

values for tomorrow. Assets can include any type of investments (stock exchange, foreign currencies, etc.). Regulatory authorities mandate financial institutions to estimate the risk of investments and make corresponding provisions to minimize the risk that an unexpected event causing severe investment loss compromises the stability of the financial system. Better risk estimations allow financial institutions to minimize the money dedicated for provisions, which can be used instead for other investments. In this paper, we assume that the risk level is 1%. This means that the amount of money allocated for provisions is what the finance institution may lose with a 1% probability. This amount of money is referred to as the value at risk (VaR). Although all the developments in this paper have been made for a 1% loss probability, any other (small) probability value can be used. For the sake of simplicity, we will subsequently refer to the historical asset values as "the samples".

2.1. Monte Carlo Method

The conventional approach is to consider that samples are not independent but rather correlated [8]. To include such dependencies in the method, we calculate the day-to-day variation of samples. Using these variations, the covariance matrix can be calculated. Dependencies are actually included by using the Cholesky matrix [9] associated with the covariance The so-called Cholesky decomposition consists of decomposing a Hermitian positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose. It is a method frequently used because of its efficiency [10]. Once the Cholesky matrix is calculated, random Gaussian (independent) samples are generated, one vector of the samples at a time. By just multiplying the Cholesky matrix by the vector of independent samples, a new vector of dependent random Gaussian variables is obtained. These samples are then transformed into log-normal samples using the exponential function. In summary, we generated Gaussian random samples, introduced historical dependence, and transformed it into log-normal samples. This process is repeated to obtain numerous samples. The components of every vector are added to obtain an estimation for the cumulative win/loss, which is the key parameter. The probability distribution of these results constitutes the estimated distribution of the win/loss for the assets in the next investment cycle (in our case, the next day).

In this paper, we consider the gain/loss distribution. Negative numbers indicate loss, so the 1% is calculated from the left of Figure 1. Other authors consider distributions where the losses are in the positive axis, and therefore they consider the percentage from the right (i.e., 99%). The detailed procedure is as follows:

1. Calculate the covariance matrix COV from the assets' series increments.
2. Calculate the Cholesky matrix CHO from the COV matrix.
3. Repeat:
   (a) Generate several $N(0,1)$ samples (Gaussian zero media $\mu = 0$ and unitary standard deviation $\sigma = 1$), one for each asset, to obtain the sample vector $\vec{v}$.
   (b) Calculate the scenario $\vec{e} = CHO \cdot \vec{v}^{\mathsf{T}}$.
   (c) Using the vector of market values $\vec{m}$, calculate the log-normal result sample: $\vec{r} = exp(\vec{e} - 1) \cdot \vec{m}$.
   (d) Calculate total win/loss value by adding all n $\vec{r}$ components: $s = \sum_{i=1}^{n} r_i$.
4. Estimate VaR by calculating 1% percentile in s (log-normal distribution).

To generate the distribution, the previous process is repeated numerous times (typically in the order of tens of millions), and the VaR percentile is estimated (in our case, 1%, as mentioned above) from the resulting histogram.

As an example, we used a historical series of 80 days for the values of the three assets. The covariance and Cholesky matrices, both of size 3×3, are listed below for clarification:

$$COV = \begin{bmatrix} 0.0001406 & 5.19894 \cdot 10^{-5} & 7.75869 \cdot 10^{-5} \\ 5.19894 \cdot 10^{-5} & 0.0001299 & 0.0001114 \\ 7.75869 \cdot 10^{-5} & 0.0001114 & 0.0002762 \end{bmatrix},$$

$$CHO = \begin{bmatrix} 0.011861013 & 0 & 0 \\ 0.004383214 & 0.010525491 & 0 \\ 0.006541336 & 0.007868523 & 0.01309934 \end{bmatrix}.$$

An example of the market values $\vec{m}$ in USD is

$$\vec{m} = \begin{bmatrix} 18440 & 17520 & 13850 \end{bmatrix}.$$

The results for the estimation of the VaR corresponding to the three assets, after one thousand iterations, are shown in Figure 1.
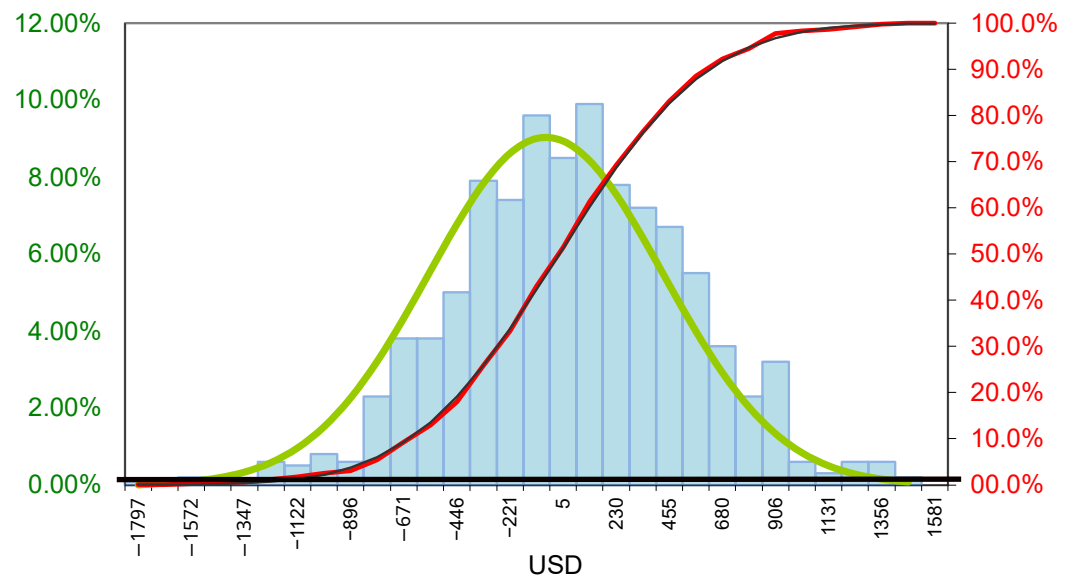

Figure 1. Monte Carlo VaR estimation histogram (blue), Gaussian fit (green) and CDF (red). The left y-axis stands for PDF and the right y-axis for CDF.

The green line represents the PDF fit using a log-normal distribution. The red line represents the CDF for the win/loss sample. The horizontal black line is the 1% threshold. The intersection of both red and black lines indicates the estimated $VaR = $ USD -1219.

2.2. Parametric Estimation

As the number of assets in an investment portfolio increases, it gets surprisingly difficult to use the Monte Carlo method to calculate the VaR. Even for an apparently small number of assets (in the order of tens), it is extremely difficult for the method to converge to a reasonable estimation since the sample space grows exponentially with the number of assets [7,11]. The alternative approach is to use a parametric estimation, which does not leverage randomness.

To estimate the VaR using the parametric method, an approximation is assumed between the input asset values and a normal distribution [12]. With this assumption, the calculus process becomes relatively straightforward, as it only requires obtaining the standard deviation of the historical series of samples.

The equation to obtain the parametric VaR for a single asset i is

$$VaR(raw)_i = V_i \cdot (exp(\sigma_i) - 1) \cdot CDF^{-1}(p), \tag{1}$$

where V is the asset value for today, σ is the standard deviation of the historical series of samples, CDF^{-1} is the inverse of the cumulative distribution for a $N(0,1)$ Gaussian distribution, and p is the level of confidence (in our case $p = 1 - 1\% = 99\%$). To include the samples' correlation, the $VaR(raw)_i$ values are arranged into a row vector. By using

the Cholesky matrix, calculated as in the Monte Carlo method, we can estimate the *VaR* by calculating the vector $\vec{w}$ as follows:

$$\vec{w} = \vec{v} \cdot CORR \cdot \vec{v}^{\mathsf{T}}, \tag{2}$$

where *CORR* is the correlation matrix of the samples, and $\vec{v}$ is the row vector of the $VaR(raw)_i$ components described in Equation (1). The *VaR* can be obtained as the square root of the accumulating $\vec{w}$ components:

$$VaR = \sqrt{\sum_{i=0}^{n} w_i^2}, \tag{3}$$

where n is the number of assets. Using the previous example with $n = 3$, we estimate $VaR = $ USD -1213, quite similar to the one obtained with the Monte Carlo approach: $VaR = $ USD -1219 (see Section 2.1). However, as this is a "mechanical" approach, it might overlook market fluctuations that are relevant for the investors ("black swans", extremely improbable but highly dangerous events). This is why it is crucial to find a method that includes randomness to calculate the *VaR* for a huge number of assets.

Nevertheless, the parametric method is still interesting because it can be used to obtain the *VaR* estimation even when the number of assets increases significantly. However, the estimation is usually worse, which means allocating more money for provisions regarding the Monte Carlo method. Other approaches have been proposed [13], some as the evolution of traditional Monte Carlo methods [14–16], or better approaches to random number generation [4,17].

3. Linear Approach to Monte Carlo Estimation

To address the computational challenge of generating numerous Gaussian random numbers, we present an approach that reduces the number of samples needed to estimate the *VaR*.

3.1. Taylor Series Approximation

Since fluctuations in asset values are typically small, one possible solution for adding randomness to the *VaR* estimation is to "linearize" the calculation scenario described above as shown in Equation (4), which is similar to a first-order Taylor series approximation [18]:

$$\vec{r} = exp(CHO \cdot \vec{v}^{\mathsf{T}} - 1) \cdot \vec{m} \approx (CHO \cdot \vec{v}^{\mathsf{T}}) \cdot \vec{m}^{\mathsf{T}}. \tag{4}$$

Given that *CHO* and $\vec{m}$ are known data, we can write Equation (4) as a linear combination of the components of $\vec{v}$, which are random variables with a $N(0,1)$ distribution as shown in Equation (5):

$$s = \sum_{i=0}^{n} r_i = \sum_{i=0}^{n} a_i \cdot v_i. \tag{5}$$

Actually, this is a sum of random variables with a $N(0,1)$ distribution. In statistical theory, it is a well known [12] that the sum of a linear combination of these types of random variables results in another Gaussian random variable $N(0,\sigma)$, whose standard deviation [11] is given by Equation (6):

$$\sigma = \sqrt{\sum_{i=0}^{n} a_i^2}, \tag{6}$$

where a_i is obtained from Equation (5).

Therefore, the Monte Carlo method to estimate *VaR* can be modified to use this linear approximation as follows:

1. Calculate the covariance matrix *COV* from the assets' series.

2. Calculate Cholesky matrix *CHO* from the *COV* matrix.
3. Calculate $\vec{a}$ using the market values vector $\vec{m}$
    $$\vec{a} = \vec{m} \cdot CHO.$$
4. Calculate the standard deviation for the $N(0, \sigma)$:
    $$\sigma = \sqrt{\sum_{i=0}^{n} a_i^2}.$$
5. Repeat:
    Generate one $N(0, \sigma)$ as a sample of the loss random variable *s*.
6. Estimate *VaR* by calculating 1% percentile in *s* (log-normal distribution).

Figure 2 shows the quasi-perfect fit of this approach for calculating loss samples. The blue line represents the parametric estimation, while the red line represents the linear approximation for five assets. Ten million samples were generated for the linear approach. The distance between both simulations can be calculated as

$$Error = \frac{\int_{-\infty}^{\infty} |PDF_{MonteCarlo} - PDF_{linear\,approximation}|}{\int_{-\infty}^{\infty} PDF_{MonteCarlo}}. \tag{7}$$

In our example, *Error* has a value of 1.5%, which we consider acceptable. We also investigated the use of a second-order Taylor approximation, reducing the error to 0.22%. However, the added complexity in calculations when using a second-order approximation inclined us to consider only the first-order approximation.
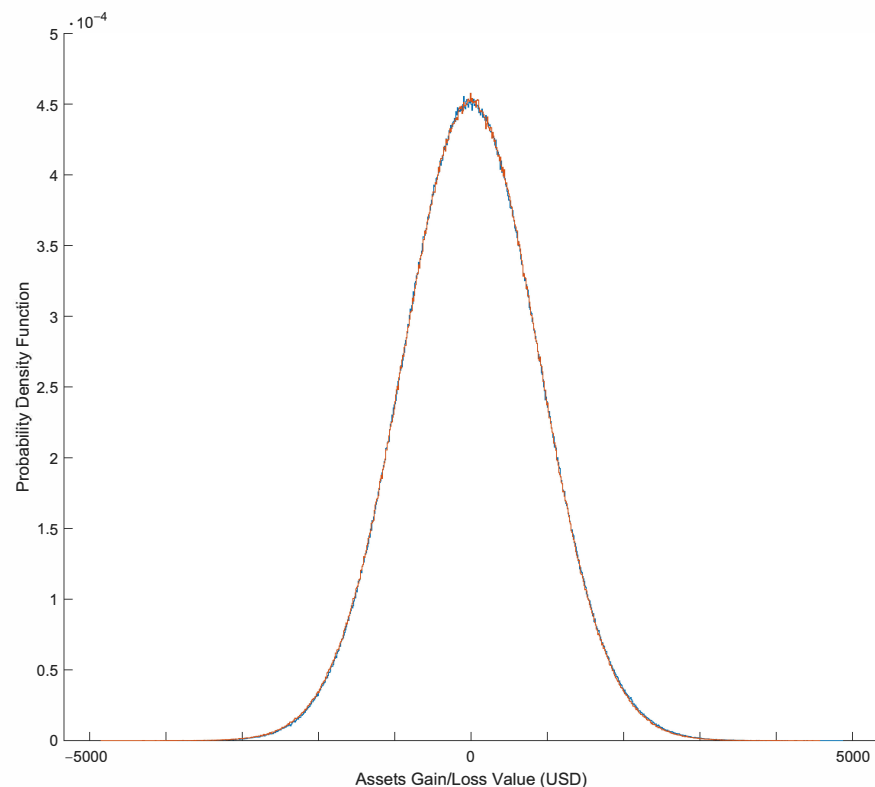

Figure 2. Gain/Loss value PDF. Comparison between the classical Monte Carlo approach (red line) and linear approximation (blue line).

While this excellent result suggests the viability of using the linear method to estimate the *VaR* for a higher number of assets, there is unfortunately a major challenge when applying this approach to hundreds of assets: Cholesky matrix calculations.

3.2. Cholesky Matrix Calculation

Both the Monte Carlo method and the linear approximation method begin with the calculation of the Cholesky matrix from the covariance matrix. The algorithm for computing the matrix components is based on an iterative process, where the covariance matrix components are used to normalize partial results [19]. This process includes a normalization step, where matrix components are used as a denominator in divisions. As it can be seen in the covariance matrix sample shown above, in Section 2.1, some matrix components can be very close to zero. If the covariance matrix is large, then division errors accumulate rapidly, causing inaccuracies in the Cholesky partial calculations and therefore turning the algorithm useless. Fortunately, this problem has already been addressed in the literature. One proposed solution is to calculate the Cholesky matrix using the LDL factorization [20], which effectively mitigates the arithmetic errors. In this paper, we used for our calculations the modifications proposed by S. H. Cheng and N. J. Higham [21].

By combining the linear approximation method and the Cheng algorithm, we were able to estimate the *VaR* for an investment portfolio consisting of 138 assets. Data were provided by a financial institution, and correspond to investments in the financial market of Peru, spanning six months. We tested the classic Monte Carlo, the parametric approximation, and the linear approximation algorithms to estimate the *VaR* of the portfolio. Tests were performed incrementally, choosing first a reduced number of assets and then incrementing the number, one by one, until reaching the full set of assets. Figure 3 represents the *VaR* calculations versus the number of assets included in the estimation. It can be seen that the parametric calculations and our linear approximation method provide similar results.
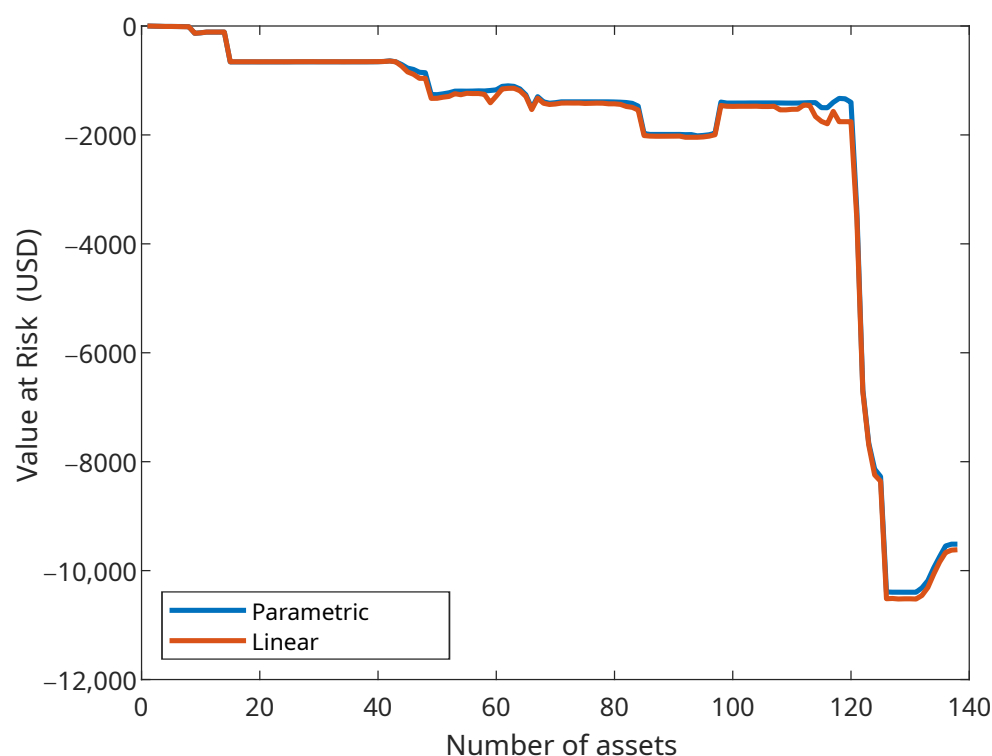

Figure 3. *VaR* evolution with number of assets.

4. Quantum Computing

As previously stated, finance institutions have concerns that random-based algorithms, like Monte Carlo, might not be able to discover less obvious scenarios when estimating the future of their investments, what they call "black swans". Consequently, they are then pushing quantum computer scientists to design and implement increasingly powerful quantum systems, with the confidence that such devices can provide better risk estimations for their business. The inherent randomness of quantum mechanics is a good argument

in favor of this approach. Based on this idea, in this section, we show how quantum computing could be used to calculate the VaR.

Quantum computers can be defined as systems designed to manipulate single particles (i.e., electrons) [22]. It is possible to program the system so that particles interact with each other in a given way, specified by the programmer. The so-called quantum state of the particles can be considered a random variable, which is measured at the end of the program (which executes in about 200 μs). The program is executed multiple times (typically 1024), and the measurements are accumulated in a histogram that is afterward sent back to the user. The discipline of quantum computing consists of designing the interactions so that the result (histogram) is relevant for a given problem. In the programming language of quantum computers, the particles are represented as so-called qubits and the manipulations and interactions as the so-called quantum gates. The representation of the program is called a quantum circuit since it resembles the aspect of classic digital circuits, although the lines do not represent wires but time.

We used the IBM Qiskit environment to develop and execute a program designed to generate the samples of assets values. Qiskit provides libraries to facilitate the development of quantum circuits.

4.1. Distribution Window Approach

For implementing the VaR calculations on a quantum computer, we used the IBM quantum experience (QE) platform and the Qiskit framework [23] to design a circuit that prepares the quantum state as described in Equation (8):

$$|0\rangle_n \mapsto |\psi\rangle = \sum_{i=0} \sqrt{p_i} |i\rangle_n. \tag{8}$$

In the literature, there are already methods to generate random samples of a distribution using Qiskit libraries. One of the proposed methods approximates the distribution function by straight segments. The number of such segments is related to the number of qubits of the computer. The more qubits, the better the approximation. However, since the number of qubits is very limited in current NISQ computers, we tried to "concentrate" the calculations within the interval of the CDF, where the VaR value is most probably located. Again, we are assuming that asset values do not have massive changes from one investment cycle to the next. Therefore, to increase the precision, we restricted the calculation of the probabilities to the window between $low = -3\sigma$ and $high = -2\sigma$. As mentioned, samples are focused on the part of the distribution where the VaR is most probably located. To further restrict the span of calculations, we used the transformation shown in Equation (9):

$$\{0, \ldots, 2^n - 1\} \ni i \mapsto \frac{high - low}{2^n - 1} \cdot i + low \in [low, high]. \tag{9}$$

Recall that the VaR with a given probability α is shown in Equation (10):

$$VaR_\alpha = inf\{x : P[X \leq x] \geq \alpha\}. \tag{10}$$

As we are only using a window of the Cumulative Distribution Function (CDF), we need to scale the parameter α. The new parameter β, which we will use within the window, is related to α as in Equation (11):

$$\beta = \alpha - \frac{1 - CDF(0, 1, 3)}{CDF(0, 1, 2) - CDF(0, 1, 3)}, \tag{11}$$

where $CDF(0, 1, x)$ is the N(0,1) CDF evaluated in $x = n \cdot \sigma$.

Using Equation (6) for the Peru portfolio of 138 assets, we obtain a value of $\sigma = 4165.26$. Given the probability of $\alpha = 1\%$, we can use Equation (11) to calculate $\beta = 47.66\%$. This transformation can be schematically seen in Figure 4.
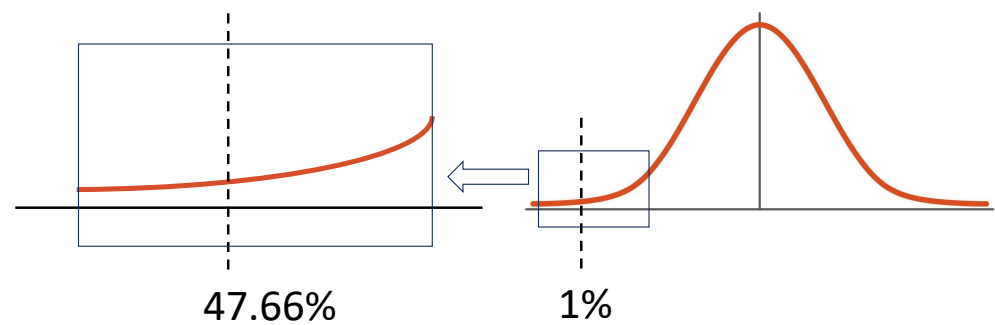
Figure 4. Window in the Gaussian distribution to focus calculations in the tail. Red line indicates the same PDF.

4.2. Statevector Simulation

The circuit for calculating the *VaR* estimation can be generated using Qiskit libraries (version 0.17.0). Qiskit uses the state preparation approach [24]. It is possible to generate a quantum circuit from a given probability distribution so that output samples are approximations to such probability distribution [25]. Qiskit libraries also allow clipping the distribution to a certain window [26]. For example, in the Peru assets, the *VaR* is estimated to be within the interval $[-3 \cdot \sigma, -2 \cdot \sigma]$. The circuit for 4 qubits is shown in Figure 5, and it includes a set of rotation and CNOT gates.
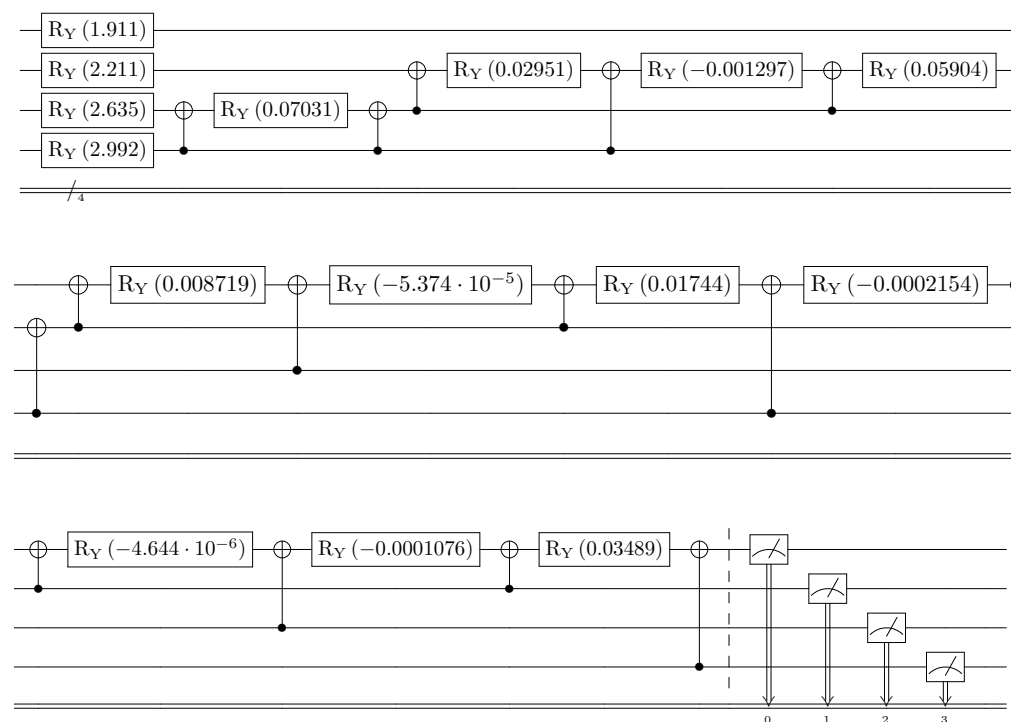

Figure 5. Sample generation circuit with 4 qubits.

By using the StatevectorSimulator (version 0.1.547), it is possible to generate samples of the risk distribution, and then the *VaR* estimation can be made by processing the results. In Figure 6, the output of the simulator is shown. As we have four qubits, the resolution is limited. The result of the "1101" state should be scaled to obtain the estimation of the $VaR = -9111.50$ USD. This value compares to the final estimation in Figure 3. With the parametric estimation used in that figure, the value at risk is $VaR = $ USD -9514.18.
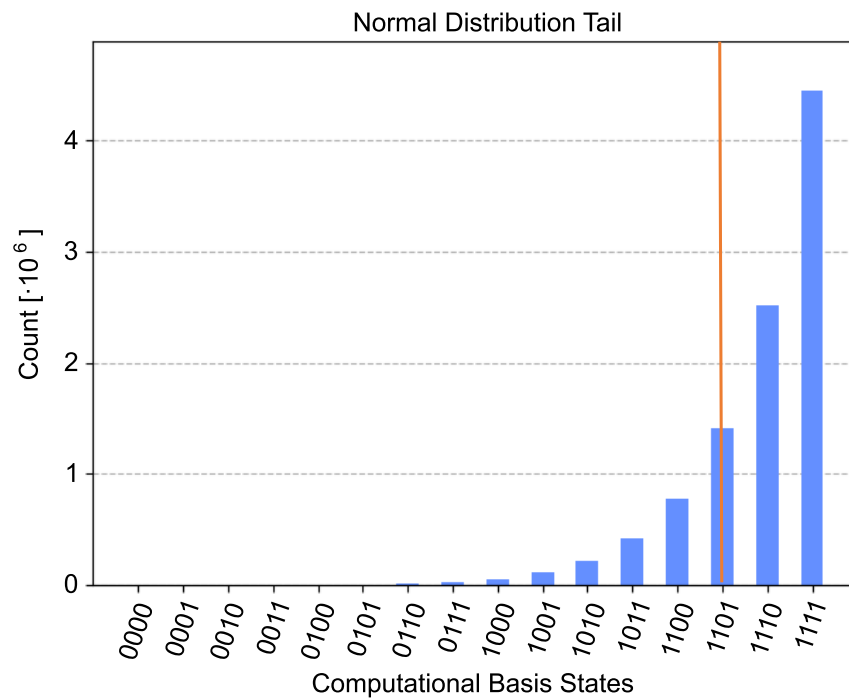
Figure 6. Output of the quantum circuit executed in the simulator. Red line indicates the *VaR* estimation.

The circuit can also be executed in a real quantum computer. The result obtained from the IBM Lagos computer is shown in Figure 7. The effect of the noise in the quantum circuit execution is clear. The histogram shape is very different from the desired Gaussian CDF tail. Actually, the *VaR*, indicated as a red line, is displaced regarding the *VaR* estimated with the simulation. This discrepancy highlights the challenges posed by the noise inherent to current NISQ computers.
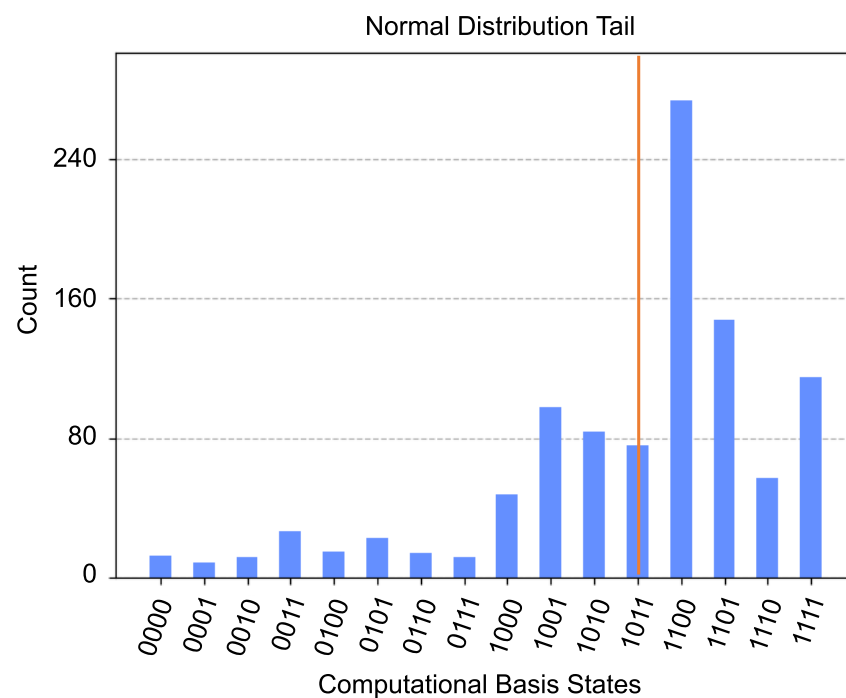

Figure 7. Output of the quantum circuit executed in the IBM Lagos quantum computer. Red line indicates the *VaR* estimation.

5. Neural Networks

Neural networks are a promising tool to mitigate the noise problem that jeopardizes the *VaR* estimation. As it is widely known, neural networks provide the perfect mechanisms for solving tasks in an alternative way [27]. Neural networks can learn from the training data to perform a specific task, and then, they can be tested with the validation data to check whether they are performing properly or not. In our case, we have the *VaR* historical series, represented in Figure 3. Therefore, we can compare the quantum computer estimation with the "real data".

In the circuit in Figure 5, the parameters that can be adjusted are the rotations. We assume that the quantum gates and the topology are essentially correct. In this scenario, fine-tuning the rotations could be enough to estimate *VaR* more precisely. Neural networks can play a pivotal role in this optimization process by helping us find the optimal rotations for a more precise *VaR* estimation in the presence of noise.

5.1. Cost Function

Neural networks learn by reducing the absolute value of the obtained cost at each epoch. Usually, this task is solved by using a default loss function provided by the machine learning library (in our case, Keras). However, in this special case, one of the main challenges is that we need to create our own cost function to establish the link between the neural network and the quantum computer output.

As previously mentioned, the neural network was designed to learn how to calculate the optimal qubit rotations to achieve the desired output. This modified design will be later executed on the quantum computer, and the estimated *VaR* will be obtained from the probability state histogram. At this point, we can return the *VaR* to the cost function to calculate the cost for that epoch.

Obtaining the desired cost is not always an easy task because having an error value of 0 does not necessarily mean that the training has been perfect or that the neural network has solved the task perfectly. The optimal scenario where the cost reaches 0 will only happen when we have seen all possible inputs and outputs for the desired task. If that is the case, training until cost is 0 would be the best decision because there will not be inputs that our neural network has not seen before. Obviously, taking all possible inputs and outputs is not the case for our task, so it would be a mistake trying to reduce the cost to exactly 0.

Anyway, even when the cost is not 0, the neural network could suffer major losses if we push the training with an excessive number of epochs. This special case is well known and referred to as overfitting, as it goes from being beneficial to a loss factor. Overfitting occurs when we train the model more than we should, and starts "memorizing" the cases instead of learning the task. Conversely, if we do not train the neural network model for enough epochs, we will be limiting its learning capacities. Therefore, the number of epochs during the training process must be a well-thought number, considering our task and design.

5.2. Layer Activation

Layer activation specifies the possible values that a neural network neuron can take. Thus, the process of choosing an appropriate layer activation plays a crucial part in the design of the neural network model. In our specific case, the range should be restricted because we are obtaining rotations for the qubits [24]. One suitable approach is to use the "ReLU" layer activation from Keras, and configuring it to saturate at the π value. Consequently, the range is precisely that of the rotation range, from $-\pi$ to π. Additionally, having a sigmoid activation layer at the output could be another approach, with the minimum mapped to $-\pi$ and the maximum mapped to π. Having a custom cost function gives us the chance to create a customizable design.

If the layer activation is wrongly chosen, the entire design may fail because the value range will not be correctly delimited, making it impossible for the neural network to learn from its executions. Furthermore, if the "ReLU" layer activation is not bounded within the

desired range, it can confuse the neural network because of the redundant rotation values occurring every 2π.

5.3. Architecture

The architecture of the implemented system was carefully designed to ensure that experiments could be efficiently carried out for different values of key parameters, such as the following:

- *Qubits*: The ability to change the number of qubits in the quantum circuit is an important parameter. Changes in the number qubits can be caused by the specific requirements of the assets or the preferences of the designer. While increasing the number of qubits can improve the precision of the system, at the same time, it also increases the computing cost. It is therefore needed to find a balance between cost and precision, which is possible through testing and result analysis. For practical reasons (mainly, quantum computer availability and calculation time), we limited the number of qubits to 5 qubits, but the method presented in this paper is applicable to any number.
- *Backend*: The flexibility of choosing a real quantum backend or a simulator without needing to change the design of the program makes development much easier. Usually, simulators are used for development, whereas the tests are typically performed in real quantum computers. Consequently, the ability to seamlessly change the backend makes the process much faster.
- *Number of assets*: When testing the developed design, we need to change the number of assets to check if the model correctly works with different scenarios. If these changes are made manually, human errors could happen, which can be avoided if we can automatically change the number of assets depending on the provided data. For training purposes, our dataset had five assets with a *VaR* of USD 1380, which was calculated with parametric estimation for a 1% percentile.
- *Grouped assets*: The number of grouped assets can easily be changed according to the preferences of the developer. While any arbitrary number can be used for this parameter, the entire design may fail if the chosen value is not appropriate. For example, if we have 100 values for each asset, it does not make sense to group them in 50 groups of 2 values because there is almost nothing to learn from each group. On the other hand, if groups contain many assets, then the number of groups is reduced, which is also not convenient for the training process. Therefore, it is important that this parameter can be freely changed if we want to find the optimal number of grouped assets for each dataset.
- *Number of layers*: The number of layers composing the model can be changed without any difficulty since the entire model is designed as an independent function that can be replaced at any moment. In this paper, we conducted different tests with two and three dense layers to conclude that the configuration with three layers provides better results.
- *Number of neurons*: Similar to the number of layers, the number of neurons in each layer can also be easily changed according to the characteristics and volume of the asset data. The number of neurons in each layer can be independently changed. In our case, we used 500 neurons at each layer, which is an amount of neurons that is high enough for learning but not too high, avoiding overfitting.

As stated above, our experiments show that using a neural network with three dense hidden layers is a suitable approach for obtaining qubit rotations in a quantum computer due to its ability to capture complex relationships and patterns in the data. Here are the main reasons for this choice:

- *Expressiveness and non-linearity*: A neural network with three dense hidden layers provides the flexibility to model complex, non-linear relationships between the input

data and the desired output (qubit rotations). This is crucial in capturing the intricate quantum mechanics involved in qubit rotations.

- *Hierarchy of features*: Three hidden layers allow for the hierarchical extraction of features from the input data. Each layer can learn increasingly abstract and higher-level representations of the qubit states, aiding in better understanding and approximating the necessary rotations.
- *Generalization and prediction*: A well-designed neural network with three dense hidden layers can generalize well to unseen data, enabling accurate predictions of the rotations needed for various output qubit configurations. This is crucial for the adaptability and performance of the quantum computer.

In summary, a 3-layer dense neural network, such as that described in Table 1 and shown in Figure 8, provides the right balance between expressiveness, parameter tuning, generalization capabilities, and computational efficiency for effectively modeling and predicting qubit rotations in a quantum computer. In the table, *nAssets* stands for the number of assets, *mGroup* for the number of grouped assets, and *kGates* the number of rotation quantum gates to adjust.

Table 1. Neural network layer architecture.

Layer Type	Input	Dense	Dense	Dense	Output
Neurons	$nAssets \cdot mGroup$	500	500	500	$kGates$

Input Layer $\in \mathbb{R}^n$ Hidden Layer $\in \mathbb{R}^{500}$ Hidden Layer $\in \mathbb{R}^{500}$ Hidden Layer $\in \mathbb{R}^{500}$ Output Layer $\in \mathbb{R}^m$
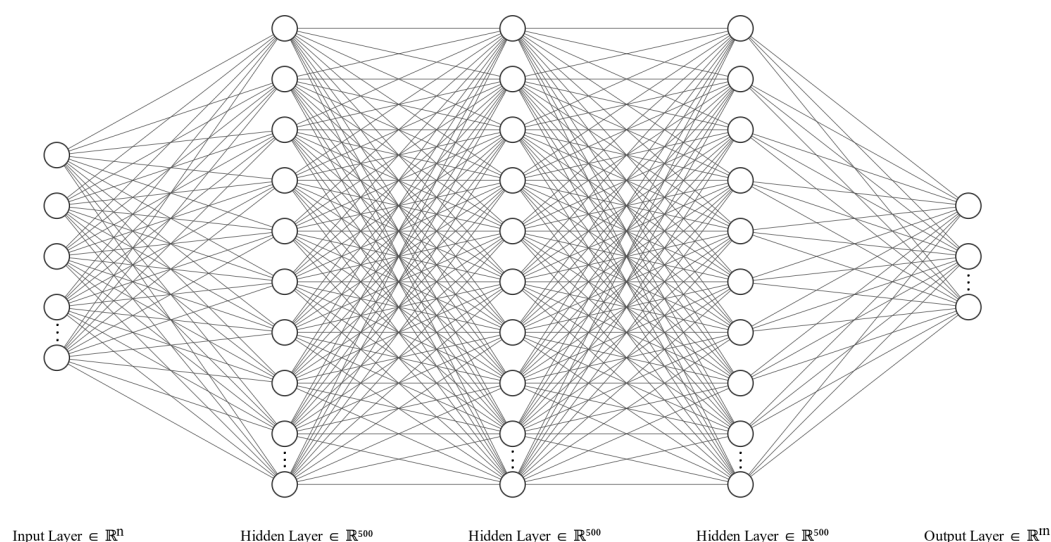
Figure 8. Neural network structure.

With these parameters in place, we developed a system that adapts to different sets of assets, number of qubits or even to real executions on quantum computers. This flexibility allows us to adjust parameter values and keep testing until we find the best possible combination for our task. The methodology used for the neural network training is shown in Figure 9.

At the top of the figure, it can be seen the first iteration of the initial epoch of the design. The first step is to calculate the historical *VaR* of each of the grouped assets. Once those groups are linked, we extract the *VaR* from the 1% percentile, which gives us the historical *VaR* of each of the asset sets.

Once we have those values, we pass the provided data to the designed program. The next step is to split the groups into training and validation. From the sets of training, we calculate the minimum *VaR* and maximum *VaR* that will be used for the distribution window range $[minVaR, maxVaR]$. The *minVaR* and *maxVaR* values correspond to the *low* and *high* parameters of Equation (9). That range is applied to Equation (11), which gives a

value of $\beta = 47.66\%$. During our tests, the decision of what percentage to use for training or validation was clear, as we had a few values for each asset. We decided to use cross validation with all values except one and try validation with the one left, as many neural network learning systems do. After splitting the input data, we passed the first group to the neural network. At that moment, the neural network had random values as it was the first time it obtained the qubit rotations, so random values for the different qubit rotations were obtained.
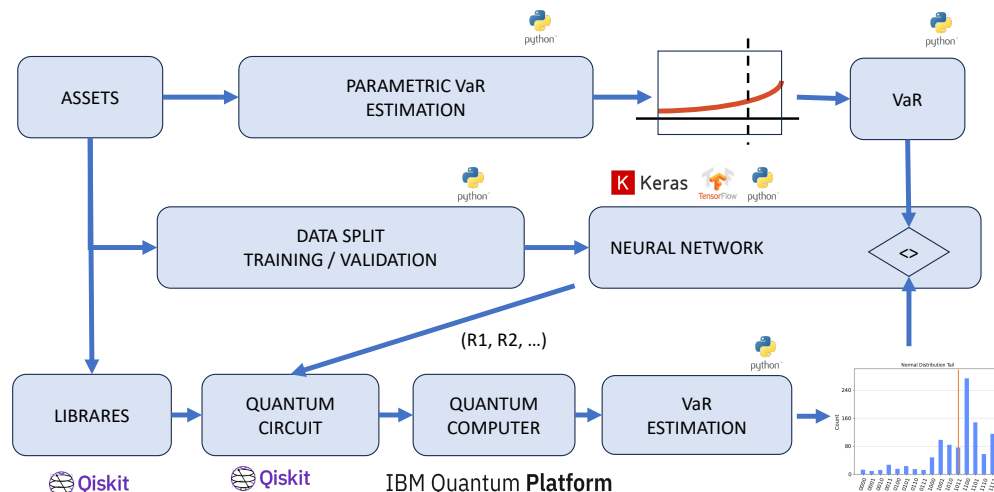

Figure 9. Execution flow of the implemented design.

With these random rotations, we edited the qubit rotations of a quantum circuit obtained with Qiskit libraries. Once we had the edited quantum circuit, it executed at the selected backend, which gave us the output histogram. With that obtained histogram, we found the β percentile (as mentioned above, in Equation (11)). With that value, we find out, on the axis that is built with the $minVaR$ and $maxVaR$, the exact VaR.

After finding the estimated VaR of the quantum circuit and the historical VaR of each group, the neural network starts the comparison between them, finding the associated cost. This cost will be used to tell the neural network whether it is learning or not. Even more, if there is a possibility to continue learning or not.

When the first group is finished the execution, it is the turn of the next group. The system performs the same steps repeatedly until it finishes the first epoch. An epoch is defined as an execution of all the different groups of inputs and outputs (x and y) provided for the neural network. After an epoch finishes, the neural network learns and tries to change the values of the different neurons to make an even better approach to the desired task. This is achieved by trying to reduce the value of the cost; if the cost is decreasing, it usually means that the system is learning correctly.

After all the epochs are completed, we can go and validate the system to check whether the task is actually an accurate solver or not and adapt the different parameters to improve it.

5.4. Execution in Real Quantum Computers

To execute the process in a real quantum computer, it is firstly needed to edit the quantum circuit obtained from the Qiskit library (in this case, a 5-qubit circuit). Once the qubit rotations are set, the circuit is sent to execution by the custom cost function (later on, it will also collect the results from the job run). As quantum computers work at the statistical level, with the possible states of the qubits, the circuit should be executed a high number of times to have a valid result, for example, 1000 to 2000 times.

It is important to have in mind that the more qubits that are used, the more complexity the circuit will have, and so the neural network would be harder to train with a low quantity of data. Furthermore, a high number of qubits increases noise effects and slows down the

training process. It is true that more qubits should increase precision, but we found that the benefits we obtained from using a limited number of qubits surpassed the disadvantages of it.

The implementation of the proposed method in a real quantum computer proves its utility to mitigate quantum noise. Figure 10 shows the obtained results. Due to the time and resource limitations, we used 5 qubits and a series of 5 assets for the estimations, but the results can be extrapolated to larger figures.
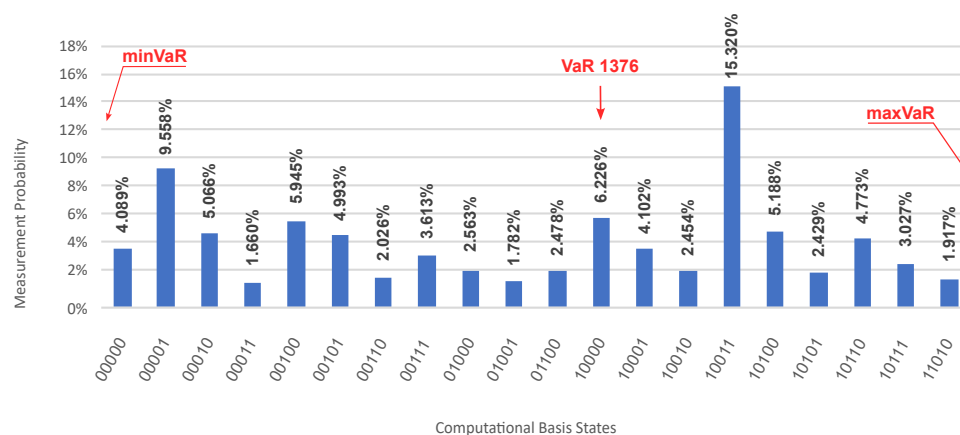

Figure 10. Real *VaR* estimation (5 qubits).

Several points may be highlighted:

1. As we used 5 qubits, we should obtain samples out of 32 values. However, in Figure 10, we only see 20. This is because Qiskit removes from the graphics the values with lower occurrences.
2. The *VaR* value is USD 1376, very close to the actual value of USD 1380 used as the target for the neural network, so the overall system seems to be well functioning.
3. The effect of the noise and the counter-effect of the rotations found by the neural network can be seen in Figure 10. In the ideal case with no noise, the distribution should be similar to the distribution tail shown in Figure 6. Although it resembles the shape, the neural network clearly modifies it. The important point is that this strange shape compensates for the noise effect.

5.5. Discussion

The results of the experiments indicate that the neural network can actually mitigate the quantum noise in the NISQ computer. The *VaR* estimated is close enough to the parametric estimation to consider it accurate enough. However, it is not exactly the desired output because quantum calculations should include (hopefully) the financial market random fluctuations. The objections to this statement are twofold. On the one hand, extensive field tests have to be performed to really make sure the quantum fluctuations somehow resemble the market complexity. The advantage of using neural networks is the flexibility to adapt the quantum circuits to perform in this way.

On the other hand, the objective of regulatory authorities is to approve the provisions made by investors. The current methodology is that the investing institution (banks) make the calculations and send them to the authorities (Central Banks). Before approving, the calculations are repeated (audited) and compared to reference calculations (for example, parametric estimations). Quantum computing is, by its own random nature, impossible to reproduce [22]. This means that, in case this approach is approved to calculate *VaR*, new protocols should be developed. Anyway, this is not only the case for this approach but a general issue with the applications of quantum computing to finance.

6. Related Work

Having presented our results, we can compare them with other works in the literature. For instance, in [28], they provide a comprehensive review of the state of the art of quantum computing for financial applications. They show that quantum algorithms are applicable for financial problems, such as *VaR* computation. They conclude that quantum Monte Carlo can speed up classical methods, but more work is required to reduce the amount of quantum resources. In our solution, we follow a different approach and obtain a reasonable result, using the resources currently available in quantum computers.

In [29], the authors present a quantum algorithm to compute risk more effectively than the Monte Carlo simulations traditionally used in classical computers. They use quantum amplitude estimation to evaluate risk measures, such as value at risk and conditional value at risk on a gate-based quantum computer. Moreover, they show how to implement this algorithm and the trade-off of the convergence rate of the algorithm and the circuit depth. The shortest possible circuit depth leads to a convergence rate of $O(M^{-2/3})$, where M is the number of samples, based on the results of [30].

However, in another work [31], they calculate the number of required qubits for these circuits, and find that they scale linearly with the number of assets, making it hard to implement this algorithm in current quantum computers when the number of assets is large. In contrast with these works, we keep using Monte Carlo simulations, focusing on the part of the distribution that is needed to calculate the *VaR* and considering how actual quantum computers behave when they use many qubits for this task. Thus, we use neural networks to mitigate the noise in the quantum circuit and obtain results comparable to those of parametric models but keeping the randomness provided by the Monte Carlo simulation.

Regarding the *VaR* computing, in [32] quantum mechanics is applied for extreme value prediction in a stock exchange. Although they do not apply quantum computing for the calculations, this work can be useful to readers not familiar with the types of distributions used to model risk prediction.

The mitigation of quantum noise is also studied in other works. In [33], they propose a protocol to estimate the average output of a noisy quantum device and demonstrate it in a 5-qubit quantum computer. In that work, they use matrix algebra to decompose the average noise of a quantum circuit. In our case, we use neural networks to adjust the rotation of the qubits and compensate for the quantum noise.

Regarding the use of neural networks, in [34], a deep quantum neural network is used to enhance the fidelity of quantum convolutional codes. As in our work, they deal with the quantum noise, although using a different approach for a different application (to enhance the fidelity of quantum states). In our case, we use traditional neural networks that take the noise data from a quantum computing, instead of the quantum neural networks they apply. In this way, we obtain a feasible solution that can be applied in current quantum computing infrastructures.

A closer work to ours is [35], where they also propose a classical artificial neural network to infer the amount of probability adjustment to mitigate quantum error. Our work differs in that we use the neural network to estimate the rotations that best compensate for the quantum noise.

7. Conclusions and Future Work

We presented a novel approach to the problem of *VaR* estimation when the number of assets is high. Monte Carlo methods are extremely difficult to apply in such cases, so parametric estimation is used instead. Using a linear approximation to the distribution of losses, we can reduce the problem to a single (or a few) random variable, which can be generated using quantum computers to consider "black swans". We presented a simulation where all the data are "compressed" in just one distribution, but it need not be the case. A *VaR* estimation problem could be divided into parts and simulated partially by real quantum computers. The results could be combined afterward to obtain the estimation.

Neural networks can be used to mitigate the noise effect in real quantum computers. Actual data were used to show that the method is feasible, and the results are close to those currently used in finance.

The validity of the fixed rotations is an open question. We proved that the neural network compensated for the noise effect in a real quantum computer, but further investigation is needed to understand the timeframe of that correction. Anyway, VaR predictions are made usually on a daily basis, so it is feasible to calculate the rotations every day. Another point is the comparison between different computers and topologies. Finally, it can be interesting to apply explainable artificial intelligence (XAI) [36] to understand how the neural network is calculating the qubit rotations to mitigate the noise, to find a simpler algorithm to compute them. The increasing size of current quantum computers makes it easier to use more and more qubits in the calculations. Nevertheless, more qubits mean more noise and further corrections. There will probably be a balance that achieves an optimum in the estimations.

Finally, as mentioned in the discussion, extensive testing with data from the field has to be performed to make sure that the application of quantum computers makes sense to predict VaR accurately. Our results suggest the accuracy of the proposed method, but much more real data are needed. The dependence of local finance markets may also be relevant for the adjustment. This topic should be addressed in future work.

Author Contributions: Conceptualization, L.d.P.; methodology, F.J.G.-A.; software, R.P.M.; validation, J.E.L.d.V.; formal analysis, L.d.P.; investigation, R.P.M.; resources, S.L.-B.; writing—original draft preparation, L.d.P., R.P.M. and F.J.G.-A.; writing—review and editing, J.E.L.d.V. and S.L.-B.; visualization, L.d.P. and R.P.M.; supervision, F.J.G.-A.; project administration, S.L.-B.; funding acquisition, S.L.-B. All authors have read and agreed to the published version of the manuscript.

Abbreviations

The following abbreviations are used in this manuscript:

CDF	Cumulative Distribution Function
CNOT	Controlled-NOT
COV	Covariance matrix
CHO	Cholesky matrix
IBM	International Business Machines
LDL	Lower triangular matrix, Diagonal matrix, Lower triagunal matrix transposed
NISQ	Noisy Intermediate Scale Quantum computing
PDF	Probability Density Function
QC	Quantum Computing
QE	Quantum Experience
ReLU	Rectified Linear Unit
USD	United States Dollar
VaR	Value at Risk
XAI	eXplainable Artificial Intelligence

References

1. Glasserman, P. *Monte Carlo Methods in Financial Engineering*; Springer: New York, NY, USA, 2004.
2. McCrary, S. *Implementing a Monte Carlo Simulation: Correlation, Skew, and Kurtosis*; Berkeley Research Group White Paper; Berkeley Research Group: Tokyo, Japan, 2015.
3. Pagès, G. *Numerical Probability: An Introduction with Applications to Finance*; Springer: Paris, France, 2018.
4. Wilson, T. Value at risk. In *Risk Management and Analysis, Volume 1: Measuring and Modelling Financial Risk*; Alexander, C., Ed.; John Wiley & Sons: Hoboken, NJ, USA, 1998.
5. Arunraj, N.; Mandal, S.; Maiti, J. Modeling uncertainty in risk assessment: An integrated approach with fuzzy set theory and Monte Carlo simulation. *Accid. Anal. Prev.* **2013**, *55*, 242–255. [CrossRef] [PubMed]
6. Preskill, J. Quantum computing in the NISQ era and beyond. *Quantum* **2018**, *2*, 79. [CrossRef]
7. Chen, N.; Hong, L.J. Monte Carlo simulation in financial engineering. In Proceedings of the 2007 Winter Simulation Conference, Washington, DC, USA, 9–12 December 2007; pp. 919–931.
8. Staum, J. Simulation in financial engineering. In Proceedings of the Winter Simulation Conference, Arlington, VA, USA, 9–12 December 2002; Volume 2, pp. 1481–1492.
9. Hazewinkel, M. Cholesky factorization. In *Encyclopedia of Mathematics*; Springer: Berlin/Heidelberg, Germany, 2001.
10. Benoit, C. Note sur une méthode de résolution des équations normales provenant de l'application de la méthode des moindres carrés à un système d'équations linéaires en nombre inférieur à celui des inconnues (Procédé du Commandant Cholesky). *Bull. Géodésique* **1924**, *2*, 66–67.
11. Springer, M.D. *The Algebra of Random Variables*; John Wiley and Sons: Hoboken, NJ, USA, 1979.
12. Ahsanullah, M.; Kibria, B.G.; Shakil, M. Normal and student's t distributions and their applications. In *Atlantis Studies in Probability and Statistics*; Springer: Berlin/Heidelberg, Germany, 2014; Volume 4.
13. Kreinovich, V.; Thach, N.N.; Trung, N.D.; Van Thanh, D. Beyond traditional probabilistic methods in economics. In *Studies in Computational Intelligence*; Springer: Berlin/Heidelberg, Germany, 2019; Volume 809.
14. Silva, M.E.d.; Barbe, T. Quasi-Monte Carlo in finance: Extending for problems of high effective dimension. *Econ. Apl.* **2005**, *9*, 577–594. [CrossRef]
15. Joe, S.; Kuo, F.Y. Constructing Sobol sequences with better two-dimensional projections. *SIAM J. Sci. Comput.* **2008**, *30*, 2635–2654. [CrossRef]
16. Joe, S.; Kuo, F.Y. Notes on generating Sobol sequences. *ACM Trans. Math. Softw.* **2008**, *29*, 49–57. [CrossRef]
17. Sobol', I.M.; Asotsky, D.; Kreinin, A.; Kucherenko, S. Construction and comparison of high-dimensional Sobol'generators. *Wilmott* **2011**, *2011*, 64–79. [CrossRef]
18. Deadman, E.; Relton, S.D. Taylor's theorem for matrix functions with applications to condition number estimation. *Linear Algebra Its Appl.* **2016**, *504*, 354–371. [CrossRef]
19. Schurman, G. The Cholesky Decomposition-Part I. 2012. Available online: http://www.appliedbusinesseconomics.com/ (accessed on 27 November 2020).
20. Golub, G.; Van Loan, C.F. *Matrix Computations*; The Johns Hopkins Univ. Press: Baltimore, ML, USA, 1996.
21. Cheng, S.H.; Higham, N.J. A modified Cholesky algorithm based on a symmetric indefinite factorization. *SIAM J. Matrix Anal. Appl.* **1998**, *19*, 1097–1110. [CrossRef]
22. Nielsen, M.A.; Chuang, I.L. *Quantum Computation and Quantum Information*; Cambridge University Press: Cambridge, UK, 2010.
23. Norlén, H. *Quantum Computing in Practice with Qiskit® and IBM Quantum Experience®: Practical Recipes for Quantum Computer Coding at the Gate and Algorithm Level with Python*; Packt Publishing Ltd.: Birmingham, UK, 2020.
24. Mottonen, M.; Vartiainen, J.J.; Bergholm, V.; Salomaa, M.M. Transformation of quantum states using uniformly controlled rotations. *arXiv* **2004**, arXiv:quant-ph/0407010.
25. Grover, L.; Rudolph, T. Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv* **2002**, arXiv:quant-ph/0208112.
26. Koch, D.; Wessing, L.; Alsing, P.M. Introduction to Coding Quantum Algorithms: A Tutorial Series Using Qiskit. *arXiv* **2019**, arXiv:1903.04359.
27. Anthony, M.; Bartlett, P.L. *Neural Network Learning: Theoretical Foundations*; Cambridge University Press: Cambridge, UK, 1999.
28. Herman, D.; Googin, C.; Liu, X.; Sun, Y.; Galda, A.; Safro, I.; Pistoia, M.; Alexeev, Y. Quantum computing for finance. *Nat. Rev. Phys.* **2023**, *5*, 450–465. [CrossRef]
29. Woerner, S.; Egger, D.J. Quantum risk analysis. *NPJ Quantum Inf.* **2019**, *5*, 15. [CrossRef]
30. Montanaro, A. Quantum speedup of Monte Carlo methods. *Proc. R. Soc. A Math. Phys. Eng. Sci.* **2015**, *471*, 20150301. [CrossRef]
31. Egger, D.J.; García Gutiérrez, R.; Mestre, J.C.; Woerner, S. Credit Risk Analysis Using Quantum Computers. *IEEE Trans. Comput.* **2021**, *70*, 2136–2145. [CrossRef]
32. Chaiboonsri, C.; Wannapan, S. Applying quantum mechanics for extreme value prediction of VaR and ES in the ASEAN stock exchange. *Economies* **2021**, *9*, 13. [CrossRef]
33. Shaib, A.; Naim, M.H.; Fouda, M.E.; Kanj, R.; Kurdahi, F. Efficient noise mitigation technique for quantum computing. *Sci. Rep.* **2023**, *13*, 3912. [CrossRef]
34. Xiao, H.; Chen, X.; Xu, J. Using a Deep Quantum Neural Network to Enhance the Fidelity of Quantum Convolutional Codes. *Appl. Sci.* **2022**, *12*, 5662. [CrossRef]

35. Kim, C.; Park, K.D.; Rhee, J.K. Quantum error mitigation with artificial neural network. *IEEE Access* **2020**, *8*, 188853–188860. [CrossRef]
36. Arrieta, A.B.; Díaz-Rodríguez, N.; Del Ser, J.; Bennetot, A.; Tabik, S.; Barbado, A.; García, S.; Gil-López, S.; Molina, D.; Benjamins, R.; et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion* **2020**, *58*, 82–115. [CrossRef]